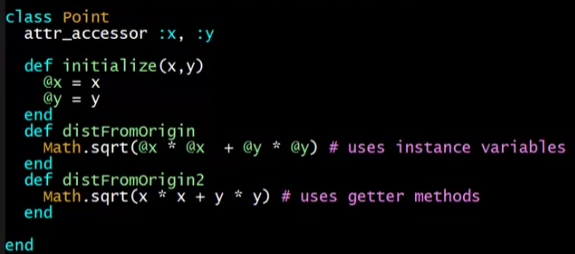
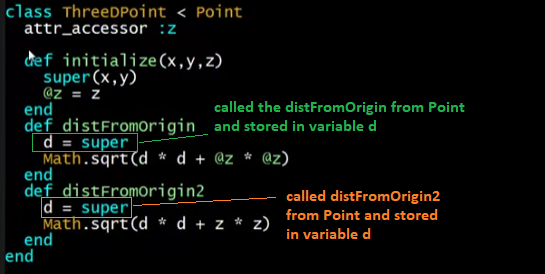
Our Point class



Our subclass ThreeDPoint (controversial)

* Abuse of subclassing
* 3d point is not really a point in 2d sense!!!



* Poor style but does overriding well
  + distFromOrigin
  + distFromOrigin2

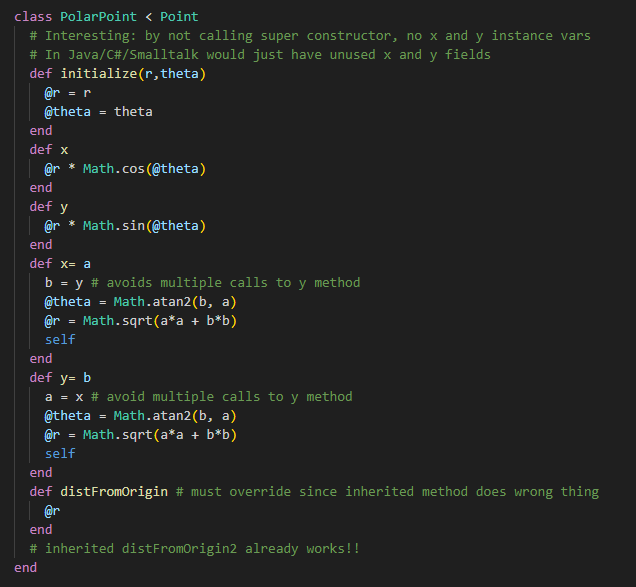
So far

* With examples so far, objects are not so different from closures
  + Multiple methods rather than just “call me”
  + Explicit instance variables rather than environment where function is defined
  + Inheritance avoids helper functions or code copying
  + “Simple” overriding just replaces methods
* But there is one big difference:

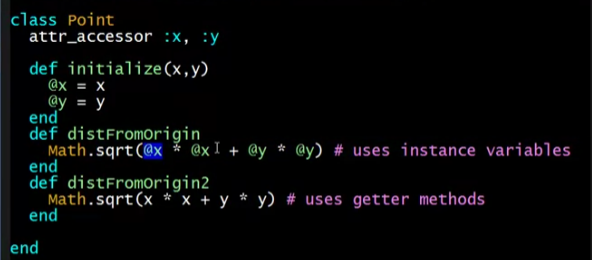
*Overriding can make a method define in the superclass call a method in the subclass*

* + The essential difference of OOP, studied carefully next lecture

Example for the big difference

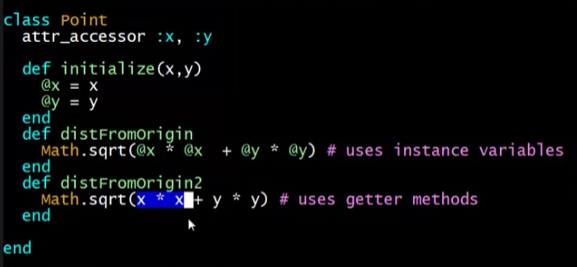


Why distFromOrigin in PolarPoint does the wrong thing?

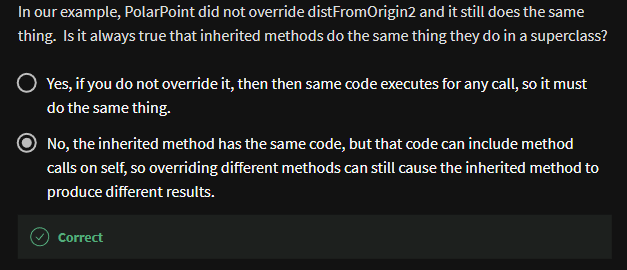


* Because it uses the instance variables x and y in the class Point! Not in the class PolarPoint

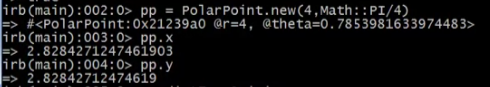
But why distFromOrigin2 does the right thing?



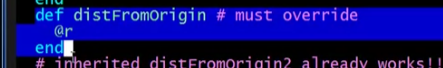
* It calls self.x and self.y
  + This means when this method is called, the class that you called is the one you are inside
    - Eg. Calling distFromOrigin2 from PolarPoint means self = PolarPoint
      * This will search for x and y inside PolarPoint
      * In our case, the computed method of x and y
    - Eg. Calling distFromOrigin2 from Point means self = Point
      * This will search for x and y inside Point
      * In our case, just the initialized variables



Using it in the REPL



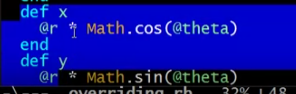




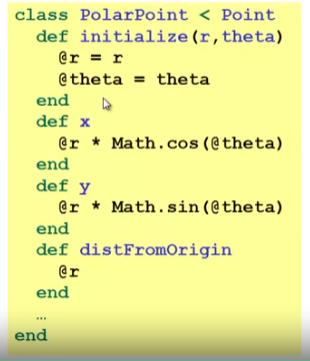




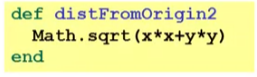




Example: Equivalent except constructor



* Also need to define x= and y=
* Key punchline:
  + distFromOrigin2 defined in Point, “already works”



* + Why? Calls to **self** are resolved in terms of the object’s class